# Prompting and Prompt Engineering

Malay Agarwal

## Contents



## Prompting

The text that is fed to LLMs as input is called the prompt and the act of providing the input is called prompting.

## Prompt Engineering

### Definition

The process of tweaking the prompt provided to an LLM so that it gives the best possible result is called prompt engineering. Some common techniques are given below.

### In-Context Learning (ICL)

In ICL, we add examples of the task we are doing in the prompt. This adds more context for the model in the prompt, allowing the model to "learn" more about the task detailed in the prompt.

### Zero-Shot Inference

For example, we might be doing semantic classification using our LLM. In that case, a prompt could be:

> Classify this review: I loved this movie!
>
> Sentiment:

This prompt works well with large LLMs but smaller LLMs might fail to follow the instruction due to their size and fewer number of features. This is also called **zero-shot inference** since our prompt has zero examples regarding what the model is expected to output.

### Few-Shot Inference

This is where ICL comes into play. By adding examples to the prompt, even a smaller LLM might be able to follow the instruction and figure out the correct output. An example of such a prompt is shown below. This is also called **one-shot inference** since we are providing a single example in the prompt:

> Classify this review: I loved this movie!
>
> Sentiment: Positive
>
> Classify this review: I don't like this chair.
>
> Sentiment:

Here, we first provide an example to the model and then ask it to figure out the output for the *I don't like this chair* review.

Sometimes, a single example won't be enough for the model, for example when the model is even smaller. We'd then add multiple examples in the prompt. This is called **few-shot inference**.

In other words:

- Larger models are good at zero-shot inference.
- For smaller models, we might need to add examples to the prompt, for few-shot inference.

## Inference Configuration Parameters

The size of the model we use for our tasks depends on the the actual tasks we want to solve and the amount of compute resources available with us.

Once we have selected a model, there are some configurations that we can play with to see if the model's performance improves. Some of them are detailed below.

## Max New Tokens

This is used to limit the maximum number of new tokens that should be generated by the model in its output. The model might output fewer tokens (for example, it predicts `<EOS>` before reaching the limit) but not more than this number.

## Greedy vs Random Sampling

Some models also give the user control over whether the model should use greedy or random sampling.

### Sample Top-K and Sample Top-P

Sample Top-K and Sample Top-P are used to limit the random sampling of a model.

A top-K value instructs the model to only consider $K$ words with the highest probabilities in its random sampling. Consider the following softmax output:

| Probability | Word |
|---|---|
| 0.20 | cake |
| 0.10 | donut |
| 0.02 | banana |
| 0.01 | apple |
| ... | ... |

If $K = 3$, the model will select one of *cake*, *donut* or *banana*. This allows the model to have variability while preventing the selection of some highly improbable words in its output.

The top-P value instructs the model to only consider words with the highest probabilities such that their cumulative probability, $p_1 + p_2 + \cdots + p_K \leq P$. For example, considering the above output, if we set $P = 0.30$, the model will only consider the words *cake* and *donut* since $0.20 + 0.10 \leq 0.30$.

### Temperature

Temperature is also another parameter used to control random sampling. It determines the shape of the probability distribution that the model calculates for the next word.

Intuitively, a higher temperature increases the randomness of the model while a lower temperature decreases the randomness of the model. This temperature is passed as a scaling factor to the final softmax layer of the decoder.

If we pick a cooler temperature ($T < 1$), the probability distribution is strongly peaked. In other words, one (or a few more) words have very high probabilities while the rest of the words have very low probabilities:

| Probability | Word |
|---|---|
| 0.001 | apple |
| 0.002 | banana |
| 0.400 | cake |
| 0.012 | donut |
| ... | ... |

Notice how *cake* has a 40% chance of being picked while other words have very small chances of being picked. The resulting text will be less random.

On the other hand, if we pick a warmer temperature ($T > 1$), the probability distribution is broader, flatter and more evenly spread over the tokens:

| Probability | Word |
|---|---|
| 0.040 | apple |
| 0.080 | banana |
| 0.150 | cake |
| 0.120 | donut |
| ... | ... |

Notice how none of the words have a clear advantage over the other words. The model generates text with a higher amount of randomness and has more variability in its output.

Clearly, when $T = 1$, the model uses the softmax output as is for random sampling.